Fall 1983

Digital Filtering with the iAPX 86/20

Robert E. Canright
University of Central Florida

DIGITAL FILTERING WITH THE iAPX 86/20

BY

ROBERT ELDON CANRIGHT, JR.
B.S., University of New Orleans, 1978
B.S.M.E., University of New Orleans, 1979

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Fall Term
1983

ABSTRACT

The iAPX 86/20 (8086 with the 8087 numeric coprocessor) is considered for digital filtering. The advantage in using the iAPX 86/20 lies in the 80-bit width of the 8087 floating-point arithmetic registers. With such large arithmetic registers, the effects of coefficient roundoff and arithmetic roundoff errors on the filter output are reduced. The price paid for the improved numerical performance is the increased time spent by the system moving data to and from memory.

The method of Knowles and Olcayto for measuring the effect of coefficient roundoff is studied in detail. This method is applied to an example filter in order to demonstrate that the iAPX 86/20 can meet filter specifications that the 8086 without the numeric coprocessor (iAPX 86/10) cannot meet.

ACKNOWLEDGMENTS

I wish to extend special thanks to the management of Martin Marietta's Engineering Computing Center for supporting this investigation; to Dr. Fred O. Simons for advising me in my graduate studies; to my mother, Julieta, for teaching me to respect education; to my wife, Jeanette, for her years of support; and to Jesus Christ, my savior, who makes all things possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

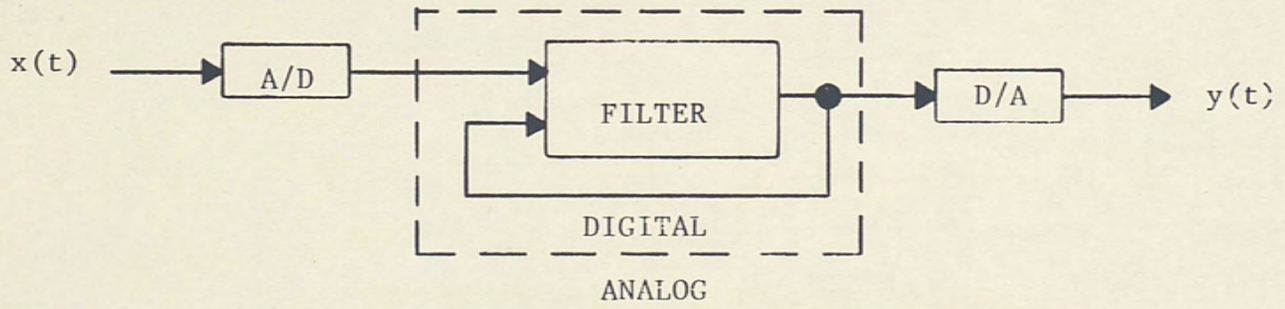
Digital filters and controllers are implemented with microprocessor systems in order to exploit some of these advantages: minimal hardware impact from algorithm changes, reduced design time compared to digital hardware or microprogramming designs, compatibility with distributed processing schemes, availability of software tools to speed design and the ability of the microprocessor to monitor signal levels for critical values or trends in order to execute different algorithms.

If software subroutines were to be used to implement floating-point arithmetic in a digital controller or filter run on a microprocessor then the resulting sample frequency would be small enough to render the scheme essentially useless for real-time processing. The advent of numeric data coprocessors like the Intel 8087 makes floating-point arithmetic a viable option. Since the floating-point addition operation of the 8087 is slower than the fixed-point addition of the 8086 and since more bus cycles are run to transfer the longer floating-point formatted numbers than the 16-bit numbers of the 8086, the iAPX 86/20 configuration (8086 + 8087) runs digital filter algorithms slower than the iAPX 86/10 configuration (8086 without coprocessors). The advantages in using the iAPX 86/20 for digital filtering or

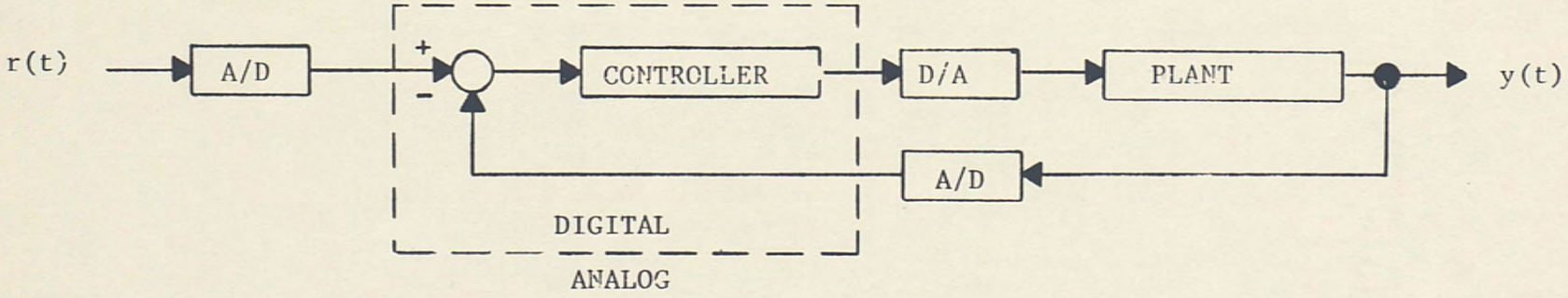
controlling would be in the use of floating-point processing with the 80-bit wide registers of the 8087.

The impact of a floating-point processor in digital filtering and control would be seen in the finite word length effects. The purpose of this investigation is to evaluate the ability of the 8087 to mitigate the adverse effects of finite precision arithmetic in digital filters implemented on the 8086.

This investigation is limited to digital filters and the techniques used herein cannot be applied directly to digital controllers due to the presence of A/D converters in the feedback loop of the digital controller [1,2]. See Figure 1.



(a.) Digital filter.



(b.) Digital controller.

Figure 1. Controllers Have Sampled Feedback

II. DIGITAL FILTER FORMS

A digital filter implements a difference equation to transform an input number sequence into an output number sequence. Recursive filters are described mathematically in the time domain by the difference equation

$$y(k) = a_0x(k) + a_1x(k-1) + \ldots + a_N(k-N) - b_1y(k-1) - b_2y(k-2) - \ldots - b_N(k-N)$$

and described mathematically in the frequency domain by the transfer function

$$H(z^{-1}) = \frac{Y(z^{-1})}{X(z^{-1})} = \frac{\sum_{i=0}^{N} a_i z^{-1}}{1 + \sum_{j=1}^{N} b_j z^{-j}} .$$

Due to arithmetic roundoff, the output is also a function of the manner in which the filter is implemented. Additional description beyond the purely mathematical is required to describe the filter implementation. The two most common methods are the signal flow graph and a modified state-space method. Care must be taken in describing a filter form with state-space

methods so as to preserve its unique structure and also distinguish between a structure and its transpose [1] (a structure and its transpose will give slightly different results). It is with signal flow graphs that the major filter forms are to be developed in this section. The purpose of this section is to describe the ability of the different filter implementations to mitigate finite word length effects and explain the choice made for this investigation.

A straightforward implementation of the transfer function, shown in Figure 2, is known as the direct form 1 [3]. It uses 2N delay elements, 2N + 1 multiplications and 2N additions for the Nth order filter. Reducing the number of delay elements not only saves space in memory, but also speeds execution by reducing the number of time consuming memory access cycles.

By introducing a dummy variable, the mathematical equations describing a filter may be manipulated to reduce the number of delay elements.

$$Y(z^{-1}) = H(z^{-1})\, X(z^{-1}) = \frac{N(z^{-1})\, X(z^{-1})}{D(z^{-1})}$$

Let

$$P(z^{-1}) = \frac{X(z^{-1})}{D(z^{-1})} = \frac{X(z^{-1})}{1 + \sum_{j=1}^{N} b_j z^{-j}}$$

then

$$Y(z^{-1}) = P(z^{-1})N(z^{-1}) = P(z^{-1}) \left[\sum_{i=0}^{N} a_i z^{-1} \right]$$

so $$p(k) = x(k) - b_1 p(k-1) - \ldots - b_N p(k-N)$$

and $$y(k) = a_0 p(k) + a_1 p(k-1) + \ldots + a_N p(k-N)$$

The signal flow graph of these last equations is shown in Figure 3 and is known as the direct form 2 [3]. The direct form 2 has the same number of additions and multiplications as the direct form 1, but has half the number of delay elements. Since the direct form 1 is used only to introduce the direct form 2, the direct form 2 is commonly referred to in the literature as the direct form. This report will henceforth follow this convention.

The direct form is known to be sensitive to the quantization of the filter coefficients [4], meaning that small errors in the filter coefficients caused by rounding the coefficients to fit the computer word length can cause large errors in the filter performance. For this reason the cascade or parallel filter forms are used to implement digital filters.

The transfer function for the cascaded filter form may be written in the form

$$H(z^{-1}) = K \prod_{i=1}^{M} \left(\frac{1 + a_{1i} z^{-1} + a_{2i} z^{-2}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}} \right)$$

which has the same number of arithmetic operations as the direct form. The cascade form has a greater signal-to-noise ratio than the direct form (roundoff errrors affect the system as additive noise) but is still inferior to the parallel form in signal-to-noise ratio [5,6,7]. Another drawback to the use of the cascaded

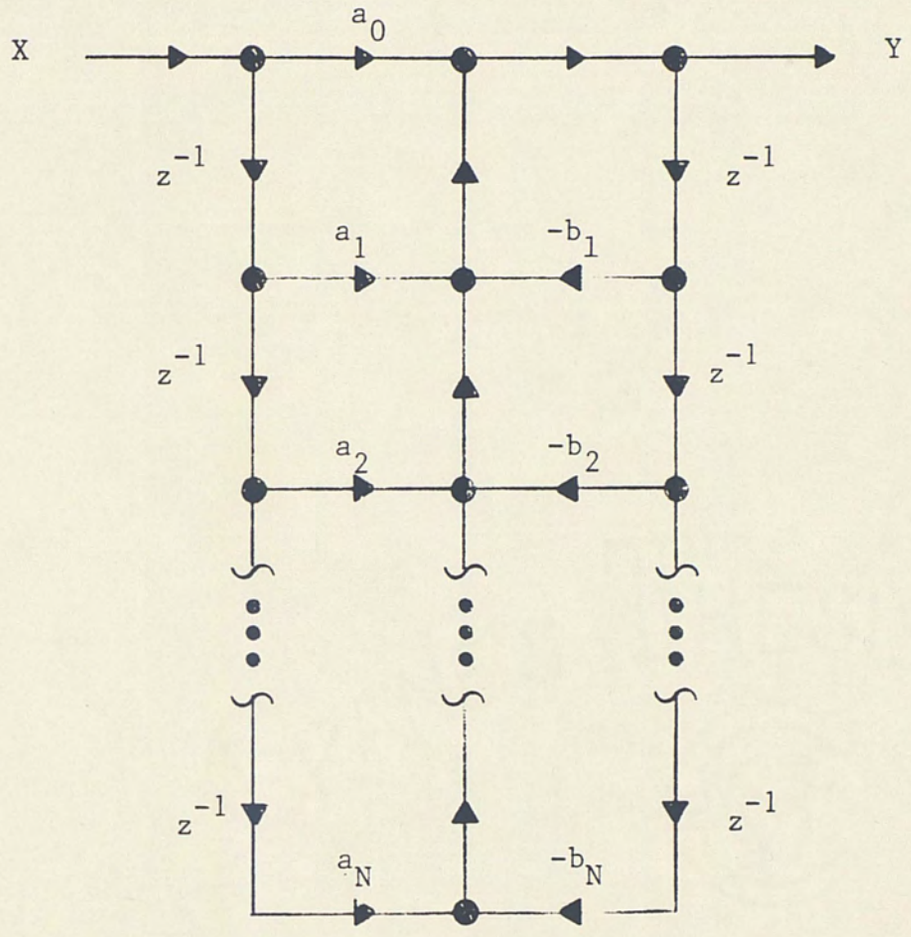


Figure 2. Flow Graph of Direct Form 1

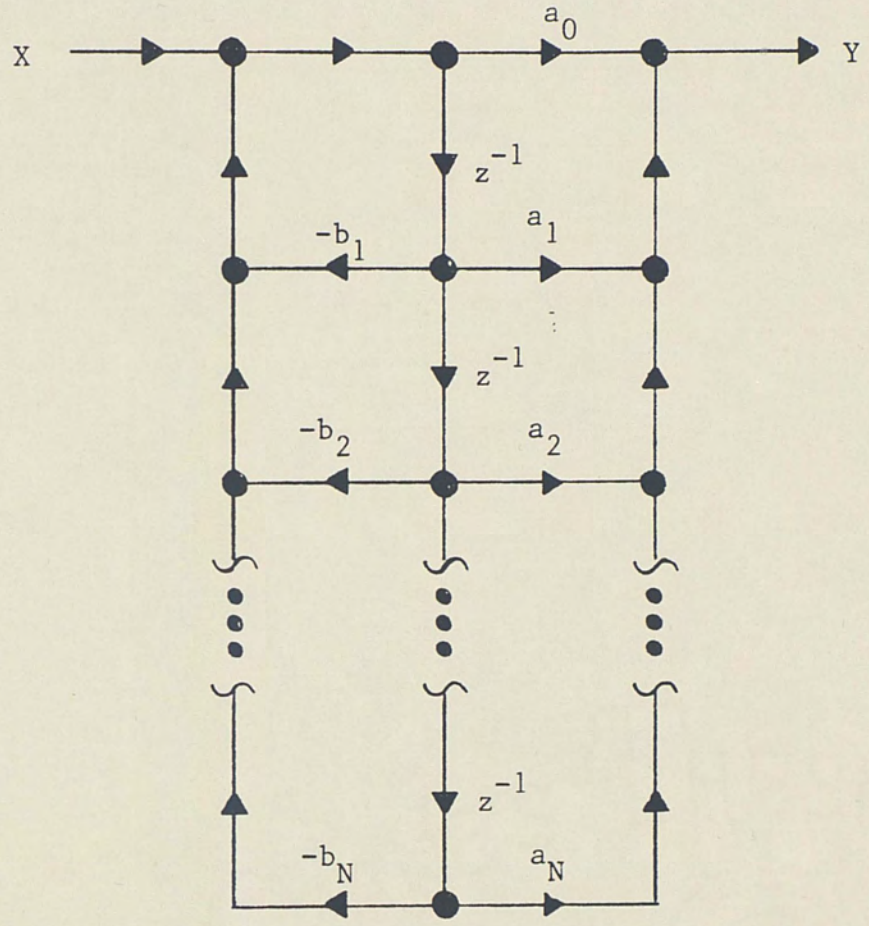


Figure 3. Flow Graph of Direct Form 2

form is the sensitivity of the cascade form to the manner in which the pole and zero pairs are combined into second order segments (the pairing problem) and also to the placement of the second order segments within the filter form (the ordering problem) [5]. The cascade form masks limit cycle oscillations when they occur and makes mathematical analysis of roundoff error and limit cycle behavior very difficult. It is for these reasons that the parallel form is used within this paper.

When a filter is implemented with fixed-point rather than floating-point arithmetic the filter coefficients must be scaled to prevent the result of an arithmetic operation from exceeding the computer word length. In order to increase the signal-to-noise ratio, scalers are placed within the second-order sections. The cascaded filter form then has the equation

$$H(z^{-1}) = \prod_{i=1}^{M} \left(\frac{a_{0i} + a_{1i} z^{-1} + a_{2i} z^{-2}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}} \right) .$$

Figure 4 shows the cascade form with scalers made up of second-order sections, each of which is implemented in the direct form. The scalers at the front and rear (input and output) have been omitted since they do not appear in the equation and can be implemented by the system's analog hardware. (Notice that the added internal scalers, a_{0i}, increase the number of multiplications.) The cascade form shown in Figure 4 has been named the

1D form by Jackson [5] to distinguish it from its transpose, called the 2D, which is shown in Figure 5.

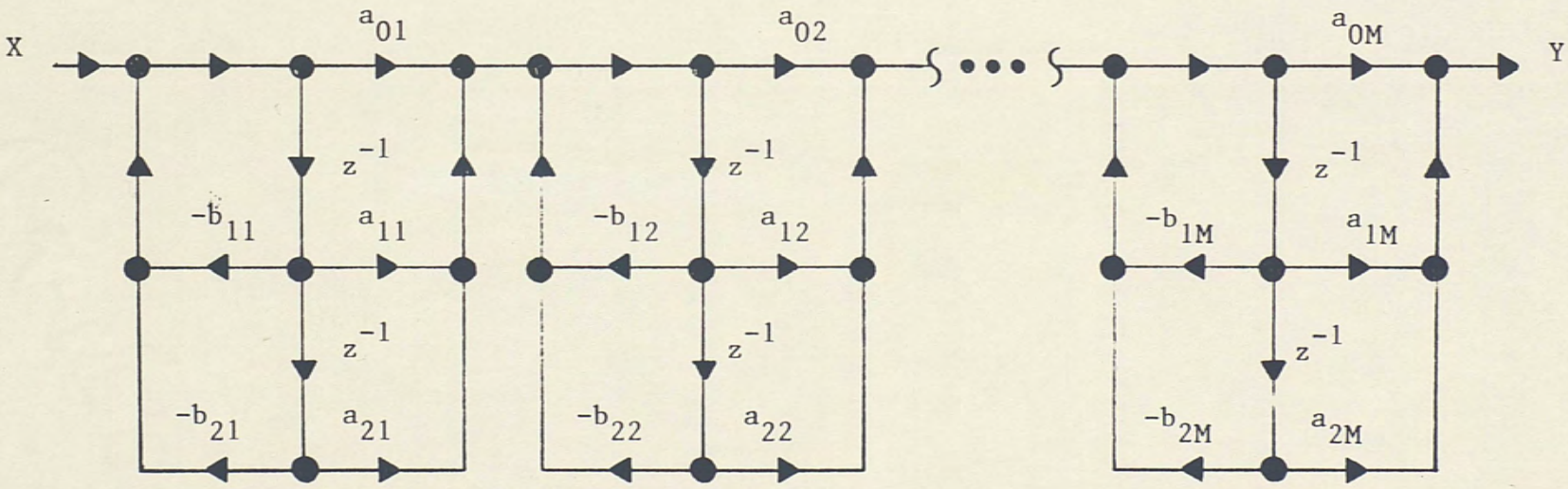


Figure 4. The 1D Cascaded Form With Internal Scalers

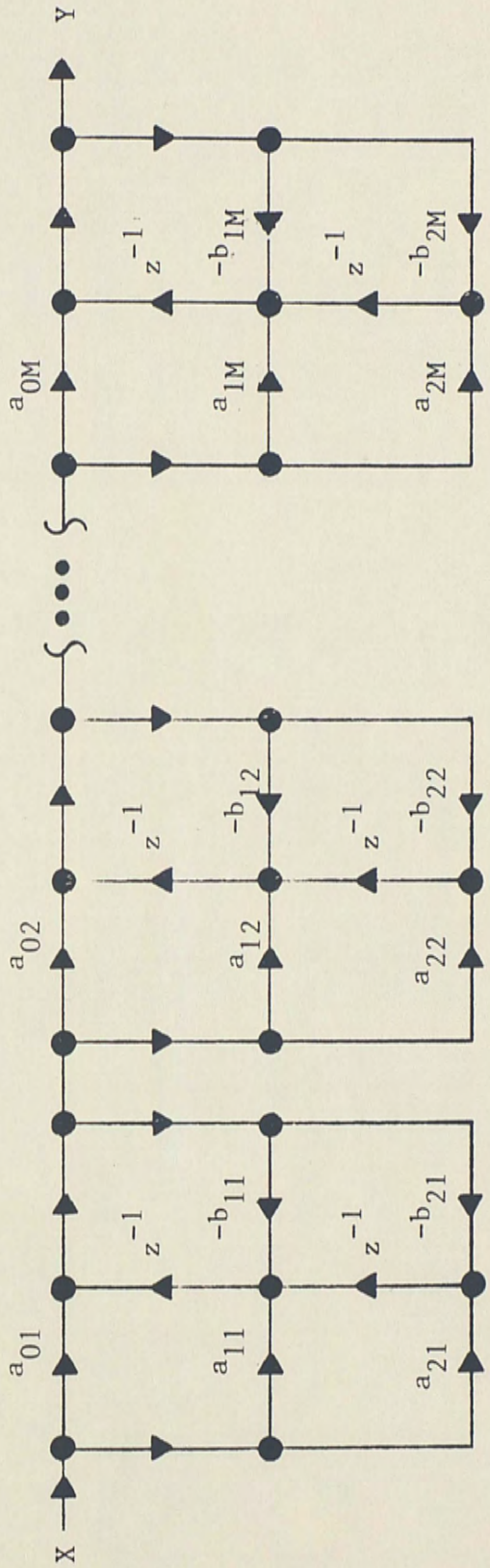


Figure 5. The 2D Cascaded Form With Internal Scalers

The transpose of a system is formed by manipulating the signal flow graph in the following manner: reverse the direction of all arrows, change all branch points to summation points, change all summation points to branch points and reverse the input and output signal names.

The parallel filter form has the equation

$$H(z^{-1}) = r + \sum_{i=1}^{M} \left(\frac{a_{0i} + a_{1i} z^{-1}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}} \right)$$

and has the same number of delays and arithmetic operations as the direct form. Figure 6 shows the parallel form made up of second-order sections without scaling multipliers since this report has its focus on floating-point processing. Jackson [5] has referred to it as the 1P form to distinguish it from its transpose, the 2P form shown in Figure 7. While there may be a large difference in the roundoff noise between the 1D and 2D forms, there is little difference between the 1P and 2P [5]. Jackson does recommend the 1P as generally exhibiting better roundoff noise than the 2P.

Having stated earlier the reasons for selecting the parallel form over the cascade form, the algorithmic implementation of the 1P and 2P forms will be examined prior to selecting one form over the other. Figure 8(a.) shows a second-order 1P section and Figure 8(b.) shows a second-order 2P section. In Figure 8(a.), s(k) is the input to the first delay branch and s(k-1) is the input to the

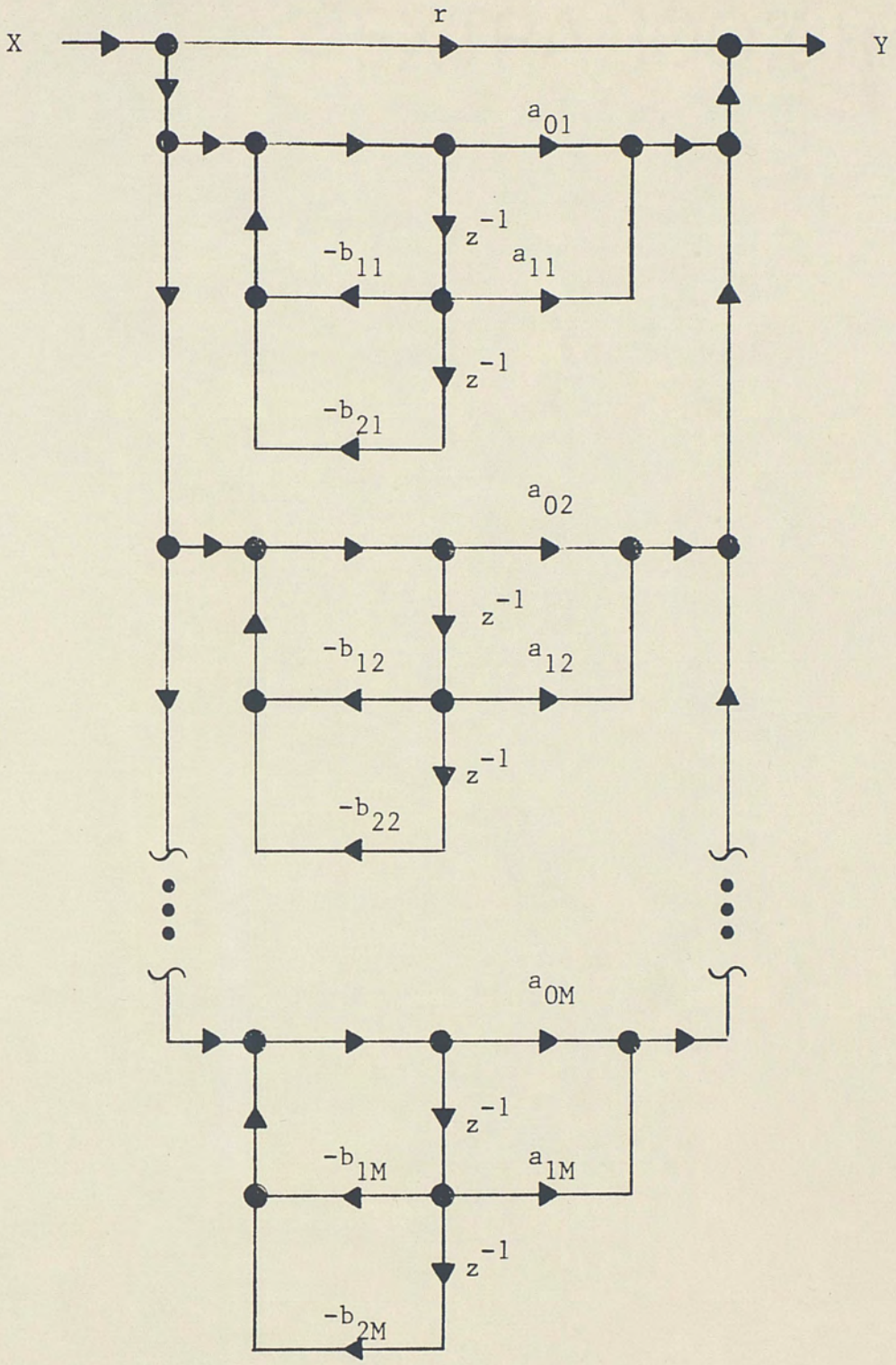


Figure 6. The 1P Parallel Form

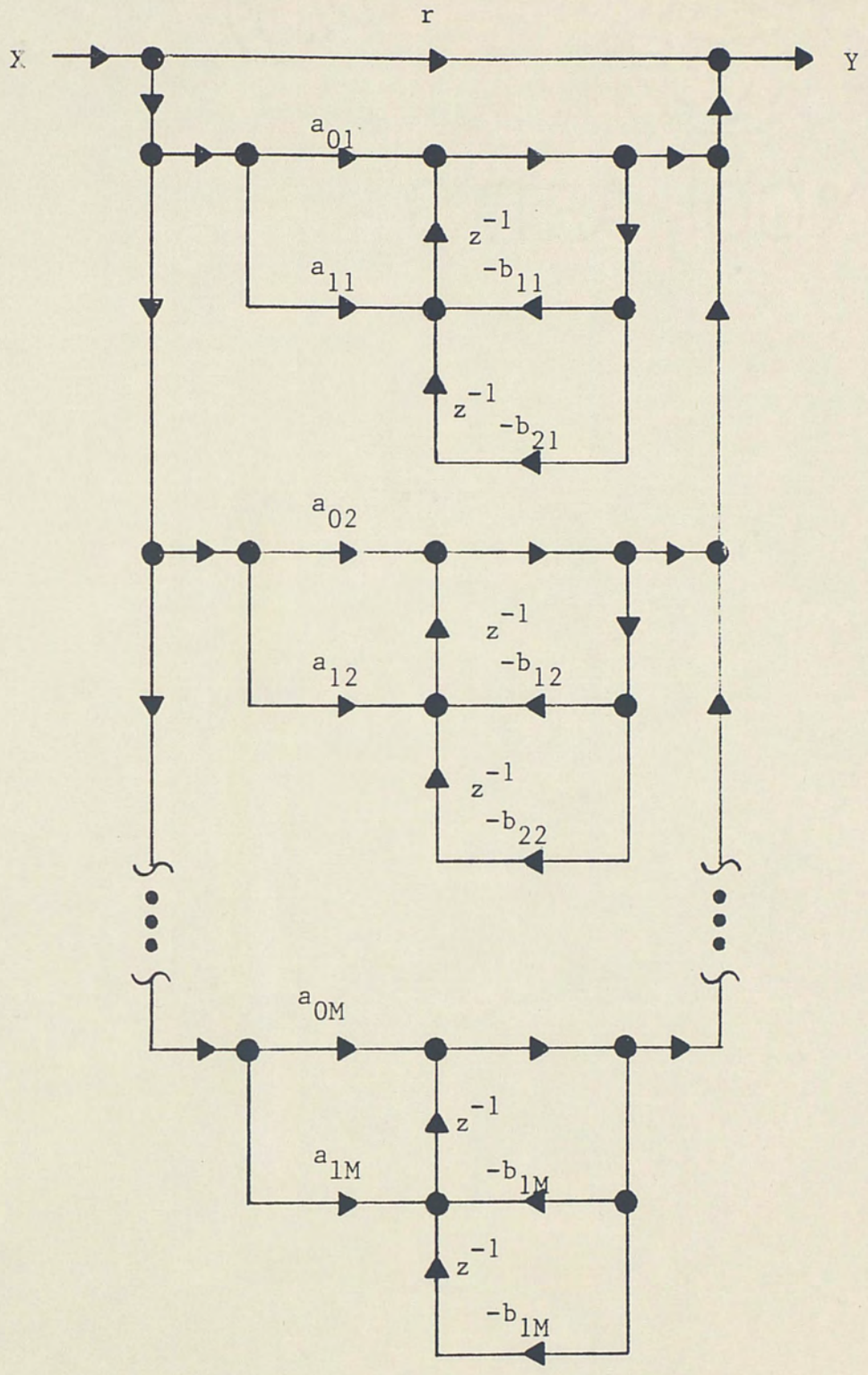
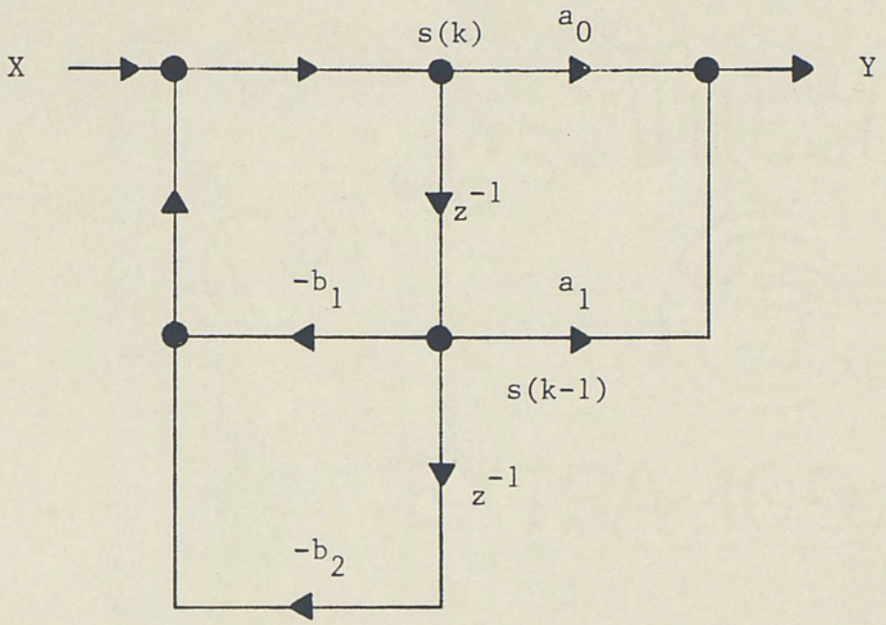
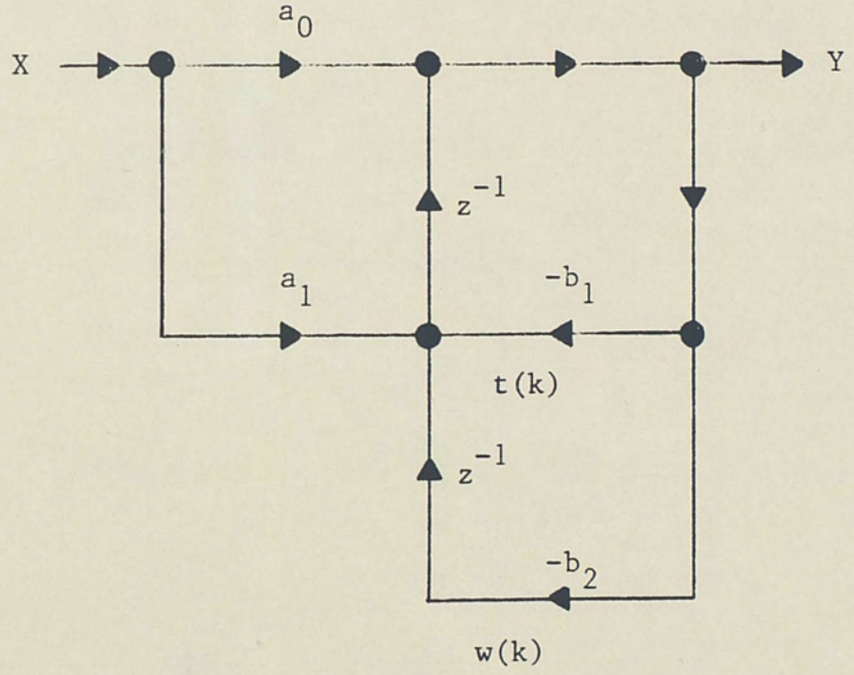


Figure 7. The 2P Parallel Form



(a.) The 1P Section.



(b.) The 2P Section.

Figure 8. Parallel Sections

second delay branch. In Figure 8(b.), w(k) is the input to the first delay branch and t(k) is the input to the second delay branch. The difference equations for the 1P and 2P sections may now be written.

$$
\begin{aligned}
\text{1P:}\quad & y(k) = a_0\, s(k) + a_1\, s(k-1) \\
& s(k) = x(k) - b_1\, s(k-1) - b_2\, s(k-2) \\
\text{2P:}\quad & y(k) = a_0\, x(k) + t(k-1) \\
& t(k) = a_1\, x(k) - b_1\, y(k) - w(k-1) \\
& w(k) = b_2\, y(k)
\end{aligned}
$$

Examination of the equation for the 1P section shows that four multiplications and three additions are required between the input and output operations - neglecting data transfer operations. (It should be remembered that the time for a floating-point addition is not small compared to a floating-point multiplication.) The processing time required between input and output is a delay that is undesirable because it is not modelled by the filter equations and can cause unacceptable filter performance. Control theory literature solves this problem by a number of techniques [1] that include modelling the time lag between input and output and eliminating the use of the current input from the equation for the current output. In practice the approach is generally to minimize the delay and test the filter performance for acceptability.

Examination of the equations for the 2P section shows that only one multiplication and one addition are required between the input and output operations. Of course, the outputs from all the second order sections must be added together to form the filter output. The variables w(k) and t(k) are computed after the output of the second order section and the time for this computation has no affect on the system if a delay loop is attached to the filter algorithm.

This chapter has explained that the choice of the 2P filter form for use in this report is based on the shorter time lag to output compared with the 1P form, the tractability of parallel forms to mathematical analysis and the greater signal-to-noise ratio of the parallel forms compared with the cascaded forms. No mention has been made of other filter forms that further reduce the sensitivity of a digital filter [4,8] to finite word length effects because they require additional arithmetic operations which slow the operation of the filter.

III. FINITE WORD LENGTH EFFECTS

The errors in digital filter performance due to limited register widths are called finite word length effects. This chapter provides a brief overview of finite word length effects. These effects are grouped into three categories: input quantization, coefficient roundoff and arithmetic roundoff.

Degradation of filter performance begins when the input signal is sampled and quantized into a digital word by the analog-to-digital converter. The smallest increment in the quantized signal is known as the quantization step size, q. When the least significant digit representing the input signal is formed by rounding, the error, e_q, is bounded by $|e_q| \leq q/2$. Assuming that e_q is a uniformly distributed random variable with zero mean, the variance of e_q is $q^2/12$. The effect of this input quantization on the output can be estimated by computing the mean-square error of the output due to the input quantization or by computing a bound on the output due to input quantization [9].

Commercially available analog-to-digital converters (ADCs) output a fixed-point binary word. The only advantage to using a floating-point ADC would be in a greater dynamic range compared to a fixed-point ADC of similar word size. Although some effort has been made to develop a floating-point ADC [10], the lack of

commercial floating-point ADCs indicates little need for such a device.

Another source of error is the rounding of the filter coefficients to accommodate register lengths. This roundoff results in shifting the pole and zero locations in the Z domain which changes the filter performance. Kaiser [6] has derived formulas for the minimum fixed-point word length necessary to insure filter stability and for the change in pole location due to the roundoff error of the filter coefficients. The interest is in predicting filter performance for a given word length to insure the word length is large enough to meet the filter specifications, so knowledge of the pole displacement is not sufficient.

Plotting the frequency response for a filter form can result in a large amount of computation since complex multiplication, addition and division operations are required along with the extraction of the magnitude of the complex valued result at each frequency for which the result is computed. Much effort has been expended in applying statistical methods to measure filter performance in order to reduce computational costs. The most notable examples are the work of Knowles and Olcayto [7], Avenhaus [11] and Crochiere [12]. The method of Knowles and Olcayto is examined in detail in Chapter V of this report and applied in Chapter VI to a floating-point filter implementation consistent with the Intel iAPX 86/20.

Another major source of error in filter performance is due to arithmetic roundoff. For the case of fixed-point arithmetic with coefficients scaled to fractions less than one there will be no overflow from multiplication but there will be an error when the product is rounded or truncated. An example will clarify how the multiplication of two binary integers will not result in overflow for scaled operands.

When the 8086 multiplies the accumulator by a byte sized operand, the lower 8-bits of the accumulator are multiplied by the 8-bit operand and a double length result is returned. A double length result is also returned when the operands are word (16-bit) sized. Suppose that the product of the scaled input, $x = 0.9$, and a scaled coefficient, $a = 0.9$, is desired and the operands are byte sized. The operands would be represented by $(0.9)\ (256) = 230.4 \cong 230$. The product is expected to be $(.81)\ (256) = 207.36 \cong 207$. The manner then in which this is achieved with the microprocessor is to drop the lower 8-bits of the result (which is the equivalent to 8 right shifts):

$$\text{1st } (230)\ (230) = 52{,}900 \ ,$$

$$\text{2nd } (52{,}900)\ (2^{-8}) = 206.64 \cong 206 \ .$$

So it is evident that the truncation which is practiced further degrades the rounded result.

The 8086 in fixed-point addition does not return double length results and overflow can result. (It is the scaling required to prevent overflow on addition that concerned Jackson [5]

in his investigation of roundoff noise.) Without the occurrence of overflow there is no error introduced to the calculations by fixed-point addition. With overflow the digital filter can exhibit oscillation [13].

Floating-point arithmetic introduces roundoff error into the calculations with both multiplication and addition. The major source of roundoff error in addition occurs when the mantissa of the smaller addend is shifted in order to match its characteristic (exponent) to that of the larger addend. This becomes most noticeable with bandpass or bandstop filters implemented in parallel second order sections where the midfrequency gains of the separate sections can vary considerably in magnitude leading to the addition (or subtraction) of numbers with great difference between them [6]. Roundoff error in floating-point digital filters has been investigated by Sandberg [14], Liu and Kaneko [15], and Kan and Aggarwal [16].

An effect due to multiplication roundoff that is unique to recursive digital filters is the limit cycle phenomenon which occurs when a zero input yields a constant nonzero output or an oscillating output. As an example consider the behavior of the first order filter defined by $y(n) = x(n) + 0.96\, y(n-1)$ shown in Table 1 with the output being rounded to two decimal places [17] and zero input. It was thought at one time that the use of floating-point arithmetic would eliminate the occurrence of limit cycles. Kaneko [17] demonstrated that large amplitude

limit cycle oscillations occur in floating-point digital filters and also derived conditions for the existence of limit cycles.

An interesting discovery by Kaneko was that no limit cycles exist in first order digital filters with binary floating-point arithmetic. This further emphasizes the desirability of implementing real poles in first order sections - of breaking filters down into sections of the smallest possible order.

Having identified the major advantage of the iAPX 86/20 over the iAPX 86/10 as the ability of floating-point arithmetic with the 80-bit wide registers of the 8087 to improve filter performance, this chapter identified the finite word length effects that could be ameliorated by the iAPX 86/20. The literature on this topic was surveyed and a method developed by Knowles and Olcayto for measuring filter output error due to coefficient roundoff was selected for investigation.

TABLE 1

A LIMIT CYCLE

n	Output	
	Before Rounding	After Rounding
0		.20
1	(.96)(.20) = .192	.19
2	(.96)(.19) = .1824	.18
3	(.96)(.18) = .1728	.17
4	(.96)(.17) = .1632	.16
5	(.96)(.16) = .1536	.15
6	(.96)(.15) = .1440	.14
7	(.96)(.14) = .1344	.13
8	(.96)(.13) = .1248	.12
9	(.96)(.12) = .1152	.12
10	(.96)(.12) = .1152	.12

IV. THE IAPX 86/20

This chapter serves to summarize the operation of the iAPX 86/20 as a digital filter. This chapter is not a tutorial on programming, nor is it a hardware reference compendium. Instead, the emphasis is on execution of the filter algorithm and the manner in which the data is formatted for floating-point processing.

In order to execute a digital filter on the iSPX 86/20, the input signal is sampled, converted to a binary representation and inputted to the 8086 CPU. (The analog input may be prefiltered by an analog circuit to prevent aliasing.) This can be performed in at least two different ways: with internal software subroutines driving the system or with external interrupts driving the system.

The digital filter driven by internal software would call a software subroutine to perform a delay loop which is used to maintain the correct input sample period. An I/O subroutine would output a control signal on one of the ports to the ADC with an OUT instruction, transfer the input word from another port to the accumulator with an IN instruction, load the input into memory with an MOV instruction, execute the filter algorithm, execute the output code and then return control to the main

program which would again call the delay subroutine. The filter algorithm would be imbedded within the I/O subroutine in order to minimize the delay between filter input and output. If separate subroutines were written for input, for the filter algorithm and for the output then time would be lost in transferring control from one subroutine to another. Furthermore, if separate subroutines were written then the I/O and filter instructions may not occupy consecutive locations in memory. This would interrupt the operation of the CPU instruction queue.

The 8086 CPU is comprised of two separate units: the execution unit (EU) and the bus interface unit (BIU). The two units operate in parallel. While the EU is executing instructions, the BIU is fetching the next instructions stored in memory. The BIU instruction queue holds up to six bytes from the instruction stream and attempts to keep that queue filled so the EU is not kept waiting for its next instruction. Additionally, all interface between the CPU and the bus occurs through the BIU. When the CPU instruction pointer takes a jump, as it usually does with program branches, the instruction queue holds the wrong instructions and the CPU must wait for an instruction fetch. This is why it is wise not to allow the program to branch between input and output if the I/O delay degrades performance (which it does for both digital filtering and control).

The digital filter driven by an external interrupt would use devices external to the CPU to maintain the sample period, trigger

the ADC and send an interrupt signal to the CPU. The CPU would then execute the I/O and filter instructions as an interrupt procedure.

The iAPX 86/20 configuration allows the 8086 CPU and the 8087 NDP (numeric data processor) to share the same instruction stream. The NDP, like the CPU, is composed of two units that operate in parallel: the numeric execution unit (NEU) and the control unit (CU). The CU maintains an instruction queue that is identical to that of the BIU. The NEU executes all instructions that involve the register stack. The pertinent NEU instructions for filtering are arithmetic and data transfer instructions.

The arithmetic registers of the NDP differ significantly from the general purpose registers of the CPU in operating as a stack. The stack consists of eight registers, each 80-bits wide having the format known as temporary real, see Figure 9(c). The operation of the stack may be described most concisely by an example. The code for a 2P filter section is listed in Figure 10, written in ASM-86 (assembly language). The operation of the stack in executing this code is shown in Figure 11. (The code illustrates the operation of the stack, not efficient use of the stack.) To speed execution, the constants b_1 and b_2 are stored as negative values in variables B1 and B2. This saves the effort of swapping registers to perform a subtraction performed in step (k) of the coded example. The 2P algorithm is repeated here for convenience.

$$y(k) = a_0\, x(k) + t(k-1)$$

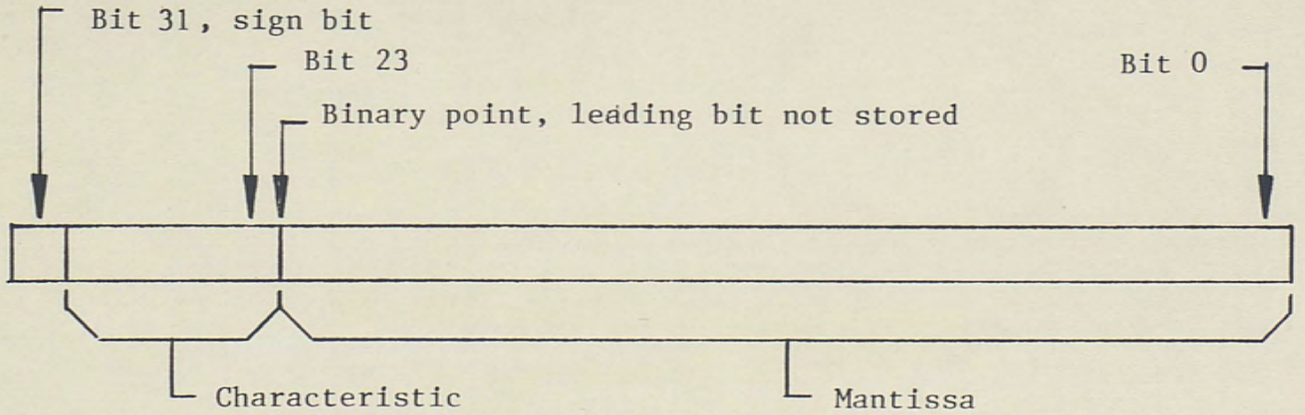
$$t(k) = a_1\, x(k) + (-b_1)y(k) + w(k-1)$$

$$w(k) = (-b_2)\, y(k)$$

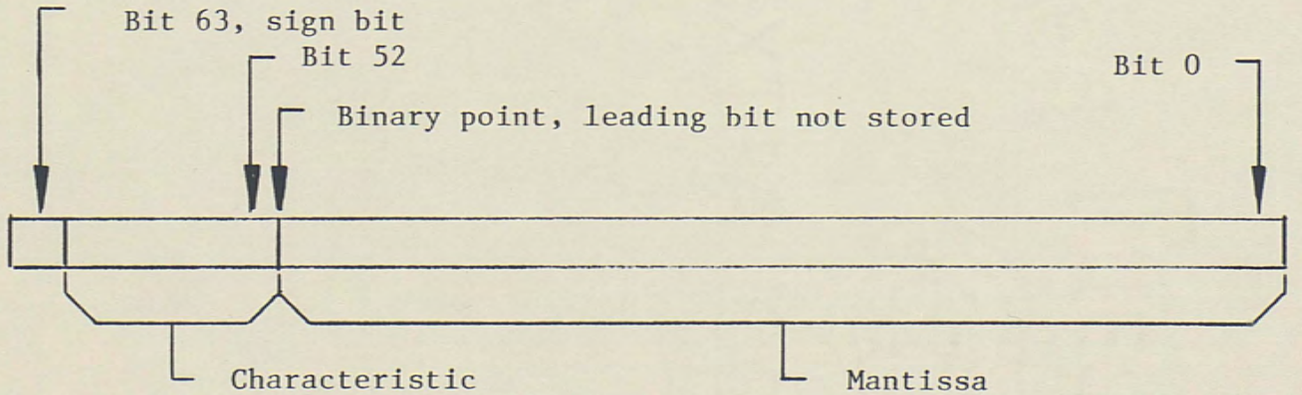
The salient features of the stack are that the registers below the stack top are designated by their position relative to the stack top and that all data transfers between the register stack and memory occur through the stack top. Since all numbers loaded into the register stack are in the temporary real format, 16-bit integer valued numbers may be operated on by 64-bit (long real) or 32-bit (short real) floating-point coefficients stored in memory. This gives the designer a good tool with which to combat coefficient roundoff errors. The 80-bit wide stack registers can virtually eliminate arithmetic roundoff errors, depending upon the precision (32, 64 or 80-bits floating-point) used to store saved values (w(k), t(k), y(k)).

This chapter described the operation of the iAPX 86/20 in digital filtering. It demonstrated how a 16-bit integer valued input can be multiplied by a high precision coefficient stored in floating-point format in memory. The usefulness of this operation is the subject of Chapter VI. This chapter has not explained assembly language programming or hardware system design. Rector and Alexy [18] and Morse [19] provide good reference to the 8086. Morgan and Waite [20] discuss the 8087 as well as the 8086. The vendor's literature [21,22,23,24] is the most important source.

Appendices A and B of this report summarize pertinent ASM-86 instructions and their time of execution.

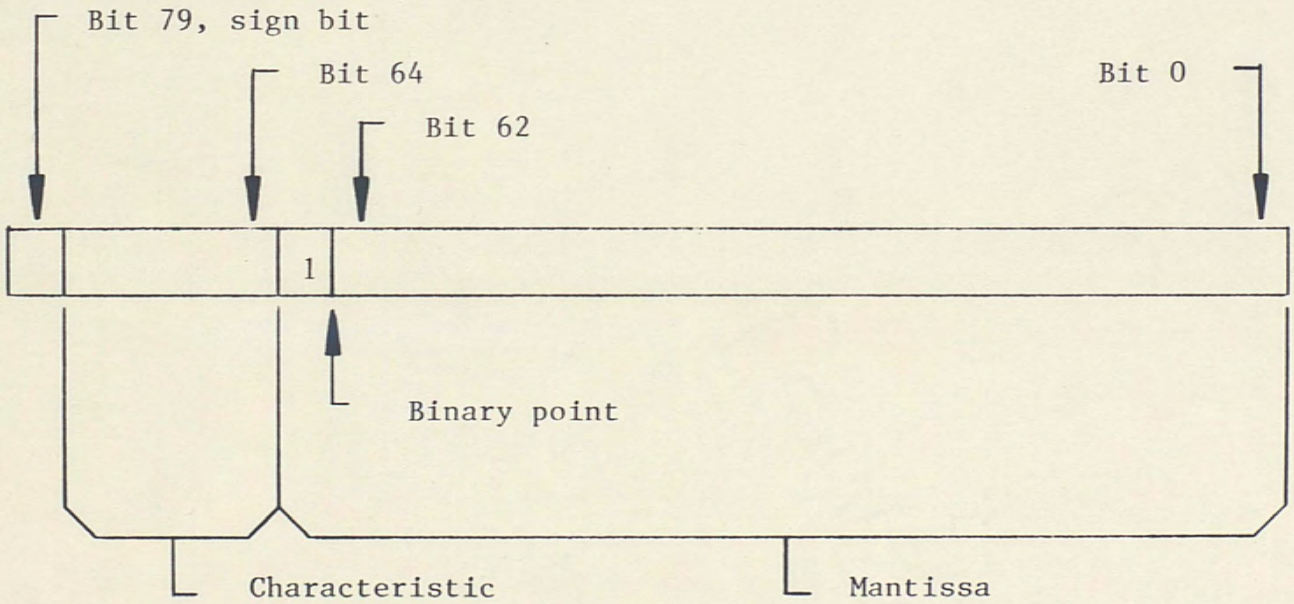


(a.) Short Real



(b.) Long real

Figure 9. Floating-Point Formats



(c.) Temporary real

Figure 9. Floating-Point Formats (Continued).

STEP	INSTRUCTION
a	FILD X
b	FLD A0
c	FMUL ST, ST (1)
d	FADD T
e	FISTP Y
f	FMUL A1
g	FADD W
h	FILD Y
i	FLD B1
j	FMUL ST, ST(1)
k	FADD ST, ST(2)
l	FSTP T
m	FLD B2
n	FMUL
o	FSTP W

Figure 10. Coded Stack Operations

ST: X (a.) FILD X	ST : AO ST(1) : X (b.) FLD AO
ST : (AO)(X) ST(1) : X (c.) FMUL ST, ST(1)	ST : (AO)(X) + T ST(1) : X (d.) FADD T
ST: X (e.) FISTP Y	ST: (X)(A1) (f.) FMUL A1
ST: ((X)(A1)+W) (g.) FADD W	ST : Y ST(1) : ((X)(A1)+W) (h.) FILD Y

Figure 11. The Stack in Operation

ST : B1 ST(1) : Y ST(2) : ((X)(A1)+W) (i.) FLD B1	ST : (B1)(Y) ST(1) : (Y) ST(2) : ((X)(A1)+W) (j.) FMUL ST, ST(1)
ST : (B1)(Y)+((X)(A1)+W) ST(1) : Y ST(2) : ((X)(A1)+W) (k.) FADD ST, ST(2)	ST : Y ST(1) : ((X)(A1)+W) (l.) FSTP T
ST : B2 ST(1) Y ST(2) : ((X)(A1)+W) (m.) FLD B2	ST : (B2)(Y) ST(1) : ((X)(A1)+W) (n.) FMUL
ST : ((X)(A1)+W) (o.) FSTP W	

Figure 11. The Stack in Operation
(Continued)

V. THE METHOD OF KNOWLES AND OLCAYTO

The method of Knowles and Olcayto for measuring the degradation of filter response due to roundoff errors in the filter coefficients yields a single number which is the mean-square error (variance) of the filter's actual output. This chapter presents a derivation of their results.

Beginning with an infinite precision filter in direct form,

$$H_\infty(z^{-1}) = \frac{A_\infty(z^{-1})}{B_\infty(z^{-1})} = \frac{\sum_{k=0}^{N} \bar{a}_k z^{-k}}{1 + \sum_{k=1}^{N} \bar{b}_k z^{-k}},$$

the ideal coefficients are replaced by rounded coefficients,

$$H(z^{-1}) = \frac{A(z^{-1})}{B(z^{-1})} = \frac{\sum_{k=0}^{N} a_k z^{-k}}{1 + \sum_{k=1}^{N} b_k z^{-k}},$$

where the ideal coefficients, $\bar{a}_k$ and $\bar{b}_k$, are related to the actual coefficients, a_k and b_k, by the relationship

$$a_k = \bar{a}_k + \alpha_k$$
$$b_k = \bar{b}_k + \beta_k$$

and the error quantities, α_k and β_k, are statistically independent, uniformly distributed and have zero mean values.

The actual filter output, y'(n), from rounded coefficients is

$$y'(n) = \sum_{k=0}^{N} a_k\, x(n-k) - \sum_{k=1}^{N} b_k\, y'(n-k) \ .$$

The infinite precision filter output, y(n), from ideal coefficients is

$$y\ (n) = \sum_{k=0}^{N} \bar{a}_k\, x(n-k) - \sum_{k=1}^{N} \bar{b}_k\, y(n-k) \ .$$

The computational error is given by

$$e(n) = y'\ (n) - y(n)$$

which after substituting y'(n) and y(n), using the definitions of e(n), a_k, b_k, and discarding second order terms (the products $\beta_k\, e(n-k)$) yields

$$e(n) = \sum_{k=0}^{N} \alpha_k\, x(n-k) - \sum_{k=1}^{N} \bar{b}_k\, e(n-k) - \sum_{k=1}^{N} \beta_k\, y(n-k) \ .$$

Applying the Z- transform to this last equation yields

$$0 = \alpha(z^{-1})\, X(z^{-1}) - \beta(z^{-1})\, Y(z^{-1}) - E(z^{-1})\, B_\infty(z^{-1})$$

where $E(z^{-1})$ is the Z- transform of e(n),

$$\alpha(z^{-1}) = \sum_{k=0}^{N} \alpha_k z^{-k} \quad \text{and}$$

$$\beta(z^{-1}) = \sum_{k=1}^{N} \beta_k z^{-k} \quad .$$

Substituting $Y(z^{-1}) = H_\infty(z^{-1})\, X(z^{-1})$ into the last equation and regrouping terms yields

$$E(z^{-1}) = \left[\frac{\alpha(z^{-1}) - \beta(z^{-1})\, H_\infty(z^{-1})}{B_\infty(z^{-1})} \right] X(z^{-1}) \quad .$$

Since $y'(n) = y(n) + e(n)$, where $y'(n)$ is the actual output and $y(n)$ is the ideal output, it follows that $Y'(z^{-1}) = Y(z^{-1}) + E(z^{-1})$. This last equation for $Y'(z^{-1})$ and the preceding equation for $E(z^{-1})$ are next applied in obtaining mean-square error of the filter output due to coefficient roundoff.

The mean-square error is given by

$$\sigma^2 = E\left[\frac{T}{2\pi} \int_0^{2\pi/T} \left| H(j\omega) - H_\infty(j\omega) \right|^2 d\omega \right]$$

where E[x] is the expectation of x (mean of x). An explanation

for this last equation is found in Appendix C. Rewriting the equation for σ^2 into a more practical form begins with the following substitutions:

$$\left| H(j\omega) - H_\infty(j\omega) \right|^2 = \left| \frac{Y'(j\omega) - Y(j\omega)}{X(j\omega)} \right|^2$$

$$= \left| \frac{E(j\omega)}{X(j\omega)} \right|^2 = \frac{E(j\omega)\, E(-j\omega)}{X(j\omega)\, X(-j\omega)} .$$

There is also a change of variables:

since $z = e^{j\omega T}$,

then $\ln(z) = j\omega T$

and taking the derivative of both sides finally gives

$$\frac{dz}{z} = jT d\omega .$$

Applying the substitution and the change of variables to the equation for σ^2 gives

$$\sigma^2 = E\left[\frac{1}{2\pi j} \oint_G \frac{E(z)\, E(z^{-1})}{X(z)\, X(z^{-1})} \frac{dz}{z} \right]$$

where G is the circle $|z| = 1$ traversed in the counterclockwise direction. Assuming a stable filter, the expectation operator may be brought inside the integration operator. Reversing the order of the operators and substituting for $E(z)/X(z)$ yields

$$\sigma^2 = \frac{1}{2\pi j} \oint_G E \left[\left(\frac{\alpha(z) - \beta(z) H_\infty(z)}{B_\infty(z)} \right) \left(\frac{\alpha(z^{-1}) - \beta(z^{-1}) H_\infty(z^{-1})}{B_\infty(z^{-1})} \right) \right] \frac{dz}{z} .$$

Since $\alpha(z)$ and $\beta(z)$ are polynomials made up of α_k and β_k terms which are independent random variables with zero mean and since the expectation of the product (the correlation) of two independent random variables with zero mean is zero, all the cross-product terms within the expectation operator go to zero. The non-zero terms occur when the product includes the square of a random variable, α_k^2 for example. Since the expectation of the square of a zero mean random variable is the variance of the random variable, the equation finally becomes

$$\sigma^2 = \left[\sum_{k=0}^{N} \sigma^2_{a_k} \right] \frac{1}{2\pi j} \oint_G \frac{1}{B_\infty(z) B_\infty(z^{-1})} \frac{dz}{z} + \left[\sum_{k=1}^{N} \sigma^2_{b_k} \right] \frac{1}{2\pi j} \oint_G \frac{H(z) H(z^{-1})}{B_\infty(z) B_\infty(z^{-1})} \frac{dz}{z} .$$

Having assumed the coefficient errors to be zero mean uniformly distributed random variables,

$$\sigma^2_{a_k} = \frac{q^2_{a_k}}{12}$$

where q_{a_k} is the quantization step size for rounding the ideal coefficient $\bar{a}_k$.

For a filter implemented in paralleled second-order sections the mean-square error may be computed from the following set of equations derived by Knowles and Olcayto. Given

$$H(z^{-1}) = \bar{r} + \sum_{k=1}^{N} \frac{\bar{a}_{0k} + \bar{a}_{1k} z^{-1}}{1 + \bar{b}_{1k} z^{-1} + \bar{b}_{2k} z^{-2}}$$

where the bar over the coefficients indicates the infinite precision coefficients, then

$$\sigma^2 = \sigma_r^2 + \sum_{k=1}^{N} \left(\sigma_{a_{0k}}^2 + \sigma_{a_{1k}}^2 \right) \frac{1}{2\pi j} \oint_G \frac{1}{B_{\infty k}(z) B_{\infty k}(z^{-1})} \frac{dz}{z}$$

$$+ \sum_{k=1}^{N} \left(\sigma_{b_{1k}}^2 + \sigma_{b_{2k}}^2 \right) \frac{1}{2\pi j} \oint_G \frac{H_{\infty k}(z) H_{\infty k}(z^{-1})}{B_{\infty k}(z) B_{\infty k}(z^{-1})} \frac{dz}{z}$$

where

$$\frac{1}{2\pi j} \oint_G \frac{1}{B_{\infty k}(z) B_{\infty k}(z^{-1})} \frac{dz}{z}$$

$$= \frac{1 + \bar{b}_{2k}}{(1 - \bar{b}_{1k} + \bar{b}_{2k})(1 + \bar{b}_{1k} + \bar{b}_{2k})(1 - \bar{b}_{2k})}$$

and

$$\frac{1}{2\pi j}\oint_G \frac{H_{\infty k}(z)\, H_{\infty k}(z^{-1})}{B_{\infty k}(z)\, B_{\infty k}(z^{-1})}\,\frac{dz}{z}$$

$$= \frac{(B_0B_2 + B_1^2)(B_0^3A_3 + B_2^3A_0) + B_0^2B_2^2(B_0A_2 + B_2A_1)}{2(B_0B_1B_2)^3}$$

where

$$B_0 = 1 + \bar{b}_{1k} + \bar{b}_{2k}$$

$$B_1 = 2(1 - \bar{b}_{2k})$$

$$B_2 = 1 - \bar{b}_{1k} + \bar{b}_{2k}$$

$$A_0 = (\bar{a}_{0k} + \bar{a}_{1k})^2$$

$$A_1 = 3\bar{a}_{0k}^2 + 2\bar{a}_{0k}\bar{a}_{1k} + 3\bar{a}_{1k}^2$$

$$A_2 = 3\bar{a}_{0k}^2 - 2\bar{a}_{0k}\bar{a}_{1k} + 3\bar{a}_{1k}^2$$

$$A_3 = (\bar{a}_{0k}^2 - \bar{a}_{1k}^2)^2 \quad .$$

At this point it is appropriate to address a concern raised by Bede Liu [9] about the method of Knowles and Olcayto. Liu pointed to the assumption made by Knowles and Olcayto that the roundoff errors are random variables. He then argued that since the coefficients are rounded once, and henceforth maintain those rounded values, the assumption that the error is random may be weak. Liu inferred that the method of Knowles and Olcayto may be inappropriate when the order of the filter is low. The crucial point to consider is the ability of the statistical model to

provide useful information on filter performance. When the coefficients are known with accuracy and precision, the roundoff errors can be calculated and are not individually random. The assumption of random error is a simplification to apply stochastic modelling. Knowles and Olcayto demonstrated by experiment the ability of their model to calculate the mean-square error for direct and parallel form filters of high order. The question of when the filter order is too low to provide meaningful results could be an interesting topic for further investigation. However, filters of low order may not be so sensitive to coefficient roundoff errors as to require analysis by the method of Knowles and Olcayto.

In demonstrating the ability of their method to calculate the mean-square error, Knowles and Olcayto did find that their method deviated from the actual mean-square error when the word size became small. For the high order parallel form filter studied by them, their method began to break down when the word size reached 8-bits. They attributed this result to the second-order error terms which were dropped to simplify the mathematical model.

The model of Knowles and Olcayto for calculating mean-square error fo the filter output focuses on the impulse response since filter specifications generally are a description of the desired properties of the transfer function.

The method of Knowles and Olcayto was examined in detail in this chapter. Its shortcomings were examined and a set of

convenient formulas to compute the mean-square error for parallel form filters was presented.

VI. AN EXAMPLE

After Knowles and Olcayto developed their method for computing the mean-square error of the filter output they applied it to a demanding filter design and compared the performance predicted by their method to the actual measured digital filter performance [11]. The filter they examined used fixed-point arithmetic. This chapter examines the same filter when implemented in paralleled second-order sections with floating-point arithmetic and with coefficients rounded to fit the Intel 8087 floating-point memory format.

The filter design examined by Knowles and Olcayto was first specified and designed by Golden and Kaiser [25]. The filter is a bandstop type with the following requirements: a -75 db minimum attenuation in the rejection band which extends from 2596 Hz to 2836 Hz and a -0.5 db maximum attenuation below 2588 Hz and above 2844 Hz. The sampling rate is 10 kHz. The digital filter is derived by applying the bilinear transform with frequency prewarping. The filter type is elliptic of twentysecondth order. The filter coefficients originally calculated by Golden and Kaiser for a parallel filter implementation did not have enough significant digits (they were also for a different type of paralleled implementation) for the error analysis intended by Knowles and Olcayto. Therefore, Knowles and Olcayto computed the coefficients for the parallel implementation to achieve a minimum of ten significant digits. These coefficients

are listed in Table 2 and correspond to the following equation:

$$H(z^{-1}) = \bar{r} + \sum_{k=1}^{11} \frac{\bar{a}_{0k} + \bar{a}_{1k} z^{-1}}{1 + \bar{b}_{1k} z^{-1} + \bar{b}_{2k} z^{-2}}$$

where the bar over the coefficients denotes infinite precision coefficients. (Actually, the infinite precision coefficients are finite precision and need only enough precision to produce significantly different results when differencing the filter outputs due to the rounded and nonrounded coefficients.)

Knowles and Olcayto applied their method to a fixed-point filter. Their results indicate that a 23-bit word length would meet the filter specifications. They then computed the actual filter responses for various coefficient word lengths and found that 19-bits was the smallest fixed-point word length that meets the specifications. The 8086 is therefore unable to meet the filter specifications with its 16-bit word lengths.

Since the short real and long real formats for storing floating-point numbers do not store the leading bit (always a one), consideration of implementing this filter with an 8087 begins with determining the number of bits needed on the right of the binary point (the number of binary places) in the binary mantissa to represent each of the infinite precision coefficients. The number of bits needed for this are listed in Table 3. The method for calculating the table is described in Appendix D. The short real format stores 23-bits and the long real 52-bits to the right

TABLE 2

FILTER COEFFICIENTS

$\bar{r}$ = 1.69887 24578

K	$\bar{a}_{1k}$	$\bar{a}_{0k}$
1	-7.46702 70246 E-4	7.31523 91358 E-4
2	9.85519 37100 E-4	7.47644 71504 E-4
3	4.55636 20191 E-3	-4.35692 04420 E-4
4	-4.88846 42202 E-3	-4.42739 24710 E-4
5	-1.33233 84702 E-2	-7.46754 71331 E-3
6	1.18780 29670 E-2	-7.67023 09337 E-3
7	2.33347 03327 E-2	4.01927 57395 E-2
8	-1.33035 41404 E-2	4.14246 00464 E-2
9	4.36615 69186 E-2	-1.43576 73940 E-1
10	-8.84472 83670 E-2	-1.47660 37472 E-1
11	-1.16760 74543 E-1	-8.60183 71841 E-1

K	$\bar{b}_{2k}$	$\bar{b}_{1k}$
1	9.98907 99167 E-1	4.28517 54740 E-1
2	9.98883 72720 E-1	1.10647 25159 E-1
3	9.95727 86927 E-1	4.31751 24685 E-1
4	9.95630 67875 E-1	1.06373 63408 E-1
5	9.88291 89882 E-1	4.41494 78013 E-1
6	9.88007 30189 E-1	9.40740 03990 E-2
7	9.66141 83349 E-1	4.66350 62848 E-1
8	9.65181 0179 E-1	6.16263 08659 E-2
9	8.74229 35455 E-1	5.18603 79640 E-1
10	8.69460 06357 E-1	-2.04564 74227 E-2
11	5.28365 04970 E-1	2.07459 21909 E-1

TABLE 3

NUMBER OF BINARY PLACES

K	$\bar{a}_{1k}$	$\bar{a}_{0k}$	$\bar{b}_{2k}$	$\bar{b}_{1k}$
1	36	36	36	35
2	37	36	36	33
3	36	35	36	35
4	36	35	36	33
5	33	36	36	35
6	33	36	36	36
7	34	35	36	35
8	33	35	35	35
9	35	34	36	36
10	36	34	36	34
11	33	36	36	34
$\bar{r}$	34			

TABLE 4
THE VALUES OF M USED TO
CALCULATE QUANTIZATION
STEP SIZE

K	$\bar{a}_{1k}$	$\bar{a}_{0k}$	$\bar{b}_{2k}$	$\bar{b}_{1k}$
1	34	34	24	25
2	33	34	24	27
3	31	35	24	25
4	31	35	24	27
5	30	31	24	25
6	30	31	24	27
7	29	28	24	25
8	30	28	24	28
9	28	26	24	24
10	27	26	24	29
11	27	24	24	26
$\bar{r}$	23			

of the binary point. The long real format would not require rounding the supposedly infinite precision coefficients and would therefore be inappropriate for analysis by the method of Knowles and Olcayto (the decimal coefficients would have to be computed again with greater precision to use the long real format).

Having selected the short real format for the rounded coefficients, the next step is to determine the quantization step size, q, for each coefficient:

$$q = 2^{-M}$$

where

$$M = 23 + P$$

and P is the absolute value of the characteristic for the floating-point binary coefficient. P is calculated by

$$P = \text{abs}(\text{int}(\log_2 x))$$

where x is an infinite precision coefficient, int(y) is the greatest integer less than or equal to y and abs(z) is the absolute value of z. The value of M for each coefficient is listed in Table 4. With the quantization step size, q, known for each coefficient, the variance (mean-square error) for each coefficient may be computed by

$$\sigma^2 = \frac{q^2}{12} .$$

The mean-square error of the filter output is now computed using the equations listed at the end of Chapter V of this report. (The Fortran code is listed in Appendix E.) The value is

$$\sigma^2 = 0.21495 \text{ E-12}$$

or $$\sigma = 0.4636 \text{ E-6} \quad .$$

To predict the filter performance from these numbers the method of Knowles and Olcayto assumes that nearly all the possible fitler deviations are bounded by plus or minus 3σ. (Their implicit assumption is that the filter output error is normally distributed with zero mean.) The test for acceptable filter performance is therefore:

$$3\sigma \leqslant \text{abs(the acceptable gain fluctuation)}.$$

The filter specification allows 0.5 db ripple in the passband. The acceptable gain fluctuation is then

$$\pm \left(\text{antilog}_{10} \left(\frac{0.5}{20} \right) - 1 \right) = \pm .059 \quad .$$

The filter specification requires -75 db minimum attenuation in the rejection band. To find the acceptable gain fluctuation for the rejection band one finds the minimum attenuation attained by the filter when the coefficients are ideal (not rounded). For the filter being investigated, Knowles and Olcayto found the minimum attenuation for the ideal filter to be -76.5 db. The allowable gain fluctuation in the rejection band is therefore:

$$\pm \left(\text{antilog}_{10} \left(\frac{-75}{20} \right) - \text{antilog}_{10} \left(\frac{-76.5}{20} \right) \right) = \pm 0.266 \text{ E-4} \quad .$$

This means that the following relation needs to hold for the filter performance to meet the specifications:

$$\sigma \leq (0.266\ E\text{-}4)/3 = 0.89\ E\text{-}5.$$

The value computed is

$$\sigma = 0.46\ E\text{-}6$$

which does satisfy the relation and indicates that the iAPX 86/20 can provide the precision required to meet the filter specifications (with coefficients using the short real format) while the iAPX 86/10 cannot meet the specifications.

If the filter under examination in this chapter were truly to be assembled then a timing analysis would have preceded the output error analysis. The filter was examined in order to demonstrate the application of the method of Knowles and Olcayto, so the timing analysis is performed in Appendix F for the sake of completeness. The result of the analysis is that the iAPX 86/20 cannot execute the filter in question with a sample rate of 10 kHz. (The upper bound on the sample rate is 341.76 Hz when the system runs on a 5 MHz clock.) The purpose of the error analysis in this chapter is to demonstrate the benefit of using the iAPX 86/20 for digital filering presuming that a microprocessor implementation is feasible.

Knowles and Olcayto in their original work examined a digital filter implemented with fixed-point arithmetic. Within this chapter the method of Knowles and Olcayto was applied to the same filter in order to demonstrate the application of the method. The result of this application indicates that the iAPX 86/20 can meet filter specifications that the iAPX 86/10 cannot meet due to coefficient quantization errors.

VII. CONCLUSION

The ability of 80-bit wide floating-point registers to reduce roundoff effects was identified as the major advantage in using the 8087 numeric coprocessor. The impact of filter realization schemes on roundoff error, various finite word length effects caused by roundoff and the operation of the iAPX 86/20 were reviewed. The method of Knowles and Olcayto for measuring output error due to coefficient roundoff was studied in detail.

Knowles and Olcayto had studied a fixed-point filter that required more than 16-bits to meet the filter specifications. This report therefore applied the method of Knowles and Olcayto to the same filter implemented with floating-point arithmetic where the coefficients were rounded to match the short real data format of the 8087. The arithmetic capabilities of the iAPX 86/20 were able to meet the filter specifications that the arithmetic capabilities of the 8086 without the numeric coprocessor could not meet. It may be concluded that the use of the numeric data coprocessor can make it possible to meet severe digital filter resolution requirements that could not be met by the 8086 alone. However, the data-transfer and arithmetic operations on the longer floating-point formatted words consume much time. They greatly reduce the bandwidth attainable by a digital filter implemented on the iAPX 86/20. For example, the

filter examined in this report requires a sample frequency far beyond the capability of the iAPX 86/20.

The major conclusion to be drawn from this research report is that the high processing overhead associated with the long floating-point data formats restricts the digital filtering application of the iAPX 86/20 to those cases that require high resolution and low bandwidth.

APPENDIX A

This appendix summarizes ASM-86 instructions used in this report. 8087 instructions begin with the letter F, 8086 instructions do not.

IN Accumulator, port
IN transfers a byte or a word from an input port to the AL register or the AX register, respectively.

FADD //source/destination, source
With no operands (//), FADD places the sum of ST (stack top) and ST(1) in ST(1) and then pops the stack. With only a source operand, FADD places the sum of the source (in memory) and the stack top in the stack top. With two operands, FADD adds the source stack register to the destination stack register.

FILD source
FILD (integer load) converts the source memory operand from its binary integer format to temporary real and pushes the result onto the 8087 stack.

FISTP destination
FISTP (integer store and pop) rounds the content of the stack top to an integer according to the RC field of the 8087 control word, transfers the result to the destination (in memory) and then pops the stack.

FLD source
FLD (load real) pushes the source operand onto the top of the register stack. The source may be another stack register or it may be any of the real data types in memory.

FMUL //source/destination, source
With no operands, FMUL (real multiply) places the product of ST (stack top) and ST(1) in ST(1) and then pops the stack. With one operand, FMUL places the product of the source memory operand and the stack top in the stack top. With two operands, FMUL places the product of the source and destination stack registers in the destination stack register.

FST destination
FST (store real) transfers the content of the stack top to the destination, which may be another stack register or a memory

operand. If the memory operand has the short or long real format, the mantissa is rounded according to the RC field of the 8087 control word.

FSTP destination
FSTP (store real and pop) operates identically to FST except that the stack is popped after the transfer. Coding FSTP ST(0) is equivalent to popping the stack with no data transfer.

MOV destination, source
MOV (move) transfers a byte or a word from the source operand to the destination operand.

OUT port, accumulator
OUT transfers a byte or a word from the AL register or the AX register, respectively, to an output port.

APPENDIX B

This appendix summarizes the typical number of clock cycles required to execute ASM-86 instructions used in this report. All floating-point memory operands are assumed to be in the short real format.

EA stands for the number of clock cycles needed to form the address of a memory operand. The instructions used in this report require only the memory displacement to be calculated, so EA = 6.

Instructions	Clocks
IN AX, PORT	10
FADD // ST, ST(i)	85
FADD memory source	105 + EA
FILD word source	50 + EA
FISTP word destination	88 + EA
FLD memory source	43 + EA
FMUL ST, ST(i)	97
FMUL memory source	118 + EA
FST memory destination	87 + EA
FSTP memory destination	89 + EA
FSTP ST(i)	20
MOV X, AX	10
MOV AX, Y	10
OUT PORT, AX	10

APPENDIX C

The implicit assumption made by Knowles and Olcayto in their work is that the filter deviation is an ergodic random process. If X is an ergodic process then [26]

$$E[X^2] = \lim_{L \to \infty} \frac{1}{2L} \int_{-L}^{L} x^2(t)dt \quad .$$

If L is not extended to infinity in the limit then the time average is an estimate of the mean-square, denoted by

$$\hat{E}[X^2] = \frac{1}{L} \int_{0}^{L} x^2(t)dt \quad .$$

This estimate is also a random variable. If the estimator is an unbiased estimator then the mean value of the estimate is equal to the true value:

$$E[X^2] = E\left[\frac{1}{L} \int_{0}^{L} x^2(t)dt\right]$$

Substituting the filter error for X, recognizing the mean-square error as the variance and applying Parseval's theorem to the integral allows us to write

$$\sigma^2 = E\left[\frac{T}{2\pi} \int_{0}^{2\pi/T} \left| H(\omega) - H_\infty(\omega) \right|^2 d\omega\right]$$

where T is the sample period of the filter.

APPENDIX D

Many base 10 floating-point numbers when transformed to base 2 require infinite precision to exactly represent the number. When the base 10 floating-point number has been rounded, the base 2 floating-point number should be rounded so that its roundoff error is less than or equal to the roundoff error of the base 10 number.

When using the method of Knowles and Olcayto it is necessary to compare the binary representations that accurately represent the filter coefficients with the binary representations of the coefficients rounded to fit the 8087 data format. This is done in order to confirm the appropriateness of the method. If the coefficients do not need rounding to fit the data format then the use of the method does not need to be considered. If the method of Knowles and Olcayto is used then the number of bits in the mantissa of the ideal coefficients should significantly exceed the number of bits in the mantissa of the data format.

Since the 8087 short and long real data formats store only the bits to the right of the binary point in the mantissa, the method of calculating the bits to the right of the binary point in the mantissa is now demonstrated.

Step 1. For a number such as (3.0123456789) (10**-4) the smallest quantization step size is

$$q = (10)^{**}(-10-4) = (10^{**}-14) \quad .$$

Setting the roundoff error of the binary representation less than or equal to the roundoff error of the decimal representation leads to

$$2^{**}M \leq 10^{**}(-14) \quad .$$

Next, the logarithm of both sides is taken and the result manipulated.

$$M \log 2 \leq -14$$

$$M \leq -14/\log 2 = -46.5$$

$$M = \text{int}(-46.5) = -47$$

where int(x) is the greatest integer less than or equal to x.

Step 2. The largest power of 2 in the number is found.

$$K = \text{int}(\log_2 (3.0123456789(10**-4))) = -12$$

Step 3. The number of bits to the right of the binary point is the mantissa, N, is found by the following equation.

$$N = \text{abs}\ (M-K)$$

$$= \text{abs}\ (-47-(-12)) = 35$$

The decimal number 0.92 shall be considered as a numerical example. Applying the method of this appendix yields N=6. The ideal binary representation of .92 is

$$0.1110\ 1011\ 10001\ ....$$

Rounding this number with N=6 yields

$$(1.110\ 110)\ (10**-1)$$

Transforming this back to base ten yields 0.921875. The error, 0.001875, is less than the roundoff error of the original decimal number, 0.005. Rounding the number to two significant digits returns the original number, 0.92.

APPENDIX E

This appendix lists Fortran IV code for implementing the equations derived by Knowles and Olcayto for the mean-square error of a digital filter implemented in paralleled second order sections. The data in Tables 2 and 4 of this report are attached to the program by a BLOCK DATA subprogram. The program was executed on a VAX computer, so REAL*8 corresponds to a double-precision real variable declaration.

To use this program for other filters implemented in parallel sections one would make these three modifications: change the DO loop counter (k = 1,11) to reflect the new number of sections, change the data in the BLOCK DATA subprogram and, if the computer is not a VAX, change the type declarations from REAL*8 to DOUBLE PRECISION and from INTEGER*2 to INTEGER.

The program may be used to analyze both fixed-point and floating-point filters.

```
C      MAIN PROGRAM NAME IS MEANER.
       INTEGER*2 MA(2,11),MB(2,11),MR
       REAL*8 SGSQA1,SGSQA2,SGSQB1,SGSQB2,N,D,VARNCE,STDDEV,
      CSUM1,SUM2,SGSQR,A(2,11),B(2,11),R
       COMMON A,B,MA,MB,R,MR
       SUM1=0.0D0
       SUM2=0.0D0
       DO 10 K=1,11
       CALL SIGMA(SGSQA1,MA(1,K))
       CALL SIGMA(SGSQA2,MA(2,K))
       CALL SIGMA(SGSQB1,MB(1,K))
       CALL SIGMA(SGSQB2,MB(2,K))
       CALL INTGL1(B(1,K),B(2,K),N)
       CALL INTGL2(A(1,K),A(2,K),B(1,K),B(2,K),D)
       SUM1=SUM1+(SGSQA1+SGSQA2)*N
       SUM2=SUM2+(SGSQB1+SGSQB2)*D
    10 CONTINUE
       CALL SIGMA(SGSQR,MR)
       VARNCE=SGSQR+SUM1+SUM2
       STDDEV=DSQRT(VARNCE)
       WRITE(6,20)STDDEV
    20 FORMAT(' STD DEVIATION=',E14.5)
       WRITE(6,30)VARNCE
    30 FORMAT(' VARNCE=',E14.5)
       STOP
       END
C
       SUBROUTINE SIGMA(OUTPUT,M)
       REAL*8 OUTPUT
       INTEGER*2 M
       OUTPUT=((1.0/(2.0**M))**2)/12.0
       RETURN
       END
C
       SUBROUTINE INTGL1(B1,B2,N)
       REAL*8 B1,B2,N
       N=(1.0+B2)/((1.0-B1+B2)*(1.0+B1+B2)*(1.0-B2))
       RETURN
       END
C
       SUBROUTINE INTGL2(AK1,AK2,BK1,BK2,D)
       REAL*8B0,B1,B2,A0,A1,A2,A3,D1,D2,D3,D4,AK1,AK2,BK1,BK2,D
       B0=1.0+BK1+BK2
       B1=2.0*(1.0-BK2)
       B2=1.0-BK1+BK2
       A0=(AK1+AK2)**2
       A1=3.0*(AK1**2)+2.0*AK1*AK2+3.0*(AK2**2)
       A2=3.0*(AK1**2)-2.0*AK1*AK2+3.0*(AK2**2)
       A3=((AK1**2)-(AK2**2))**2
       D1=B0*B2+B1*B1
```

```
      D2=(B0**3)*A3+(B2**3)*A0
      D3=B0*A2+B2*A1
      D4=B0*B1*B2
      D=(D1*D2+B0*B0*B2*B2*D3)/(2.0*(D4**3))
      RETURN
      END
C
      BLOCK DATA
      INTEGER*2 MA(2,11),MB(2,11),MR
      REAL*8 A(2,11),B(2,11),R
      COMMON A,B,MA,MB,R,MR
      DATA R/1.6988724578D0/,MR/23/,
     CA(2,1)/-7.4670270246D-4/,A(1,1)/7.3152391358D-4/
      DATA A(2,2)/9.8551937100D-4/,A(1,2)/7.4764471504D-4/,
     CA(2,3)/4.5563620191D-3/,A(1,3)/-4.3569204420D-4/
      DATA A(2,4)/-4.8884642202D-3/,A(1,4)/-4.4273924710D-4/,
     CA(2,5)/-1.3323384700-2/,A(1,5)/-7.4675471331D-3/
      DATA A(2,6)/1.1878029670D-2/,A(1,6)/-7.6702309337D-3/,
     CA(2,7)/2.3334703327D-2/,A(1,7)/4.0192757395D-2/
      DATA A(2,8)/-1.3303541404D-2/,A(1,8)/4.1424600464D-2/,
     CA(2,9)/4.3661569186D-2/,A(1,9)/-1.4357673940D-1/
      DATA A(2,10)/-8.8447283670D-2/,A(1,10)/-1.4766037472D-1/,
     CA(2,11)/-1.1676074543D-1/,A(1,11)/-8.6018371841D-1/
      DATA B(2,1)/9.9890799167D-1/,B(1,1)/4.2851754740D-1/,
     CA(2,2)/9.9888372720D-1/,B(1,2)/1.1064725159D-1/
      DATA B(2,3)/9.9572786927D-1/,B(1,3)/4.3175124685D-1/,
     CB(2,4)/9.9563067875D-1/,B(1,4)/1.06373408D-1/
      DATA B(2,5)/9.8829189882D-1/,B(1,5)/4.4149478013D-1/,
     CB(2,6)/9.8800730189D-1/,B(1,6)/9.4074003990D-2/
      DATA B(2,7)/9.6614183349D-1/,B(1,7)/4.6635062848D-1/,
     CB(2,8)/9.65181079D-1/,B(1,8)/6.1626308659D-2/
      DATA B(2,9)/8.7422935455D-1/,B(1,9)/5.1860379640D-1/,
     CB(2,10)/8.6946006357D-1/,B(1,10)/-2.0456474227D-2/
      DATA B(2,11)/5.2836504970D-1/,B(1,11)/2.0745921909D-1/,
     CMA(2,1)/34/,MA(2,2)/33/,(MA(2,K),K=3,4)/2*31/
      DATA (MA(2,K),K=5,6)/2*30/,MA(2,7)/29/,MA(2,8)/30/,
     CMA(2,9)/28/,(MA(2,K),K=10,11)/2*27/,(MA(1,K),K=1,2)/2*34/
      DATA(MA(1,K),K=3,4)/2*35/,(MA(1,K),K=5,6)/2*31/,
     C(MA(1,K),K=7,8)/2*28/,(MA(1,K),K=9,10)/2*26/,MA(1,11)/24/
      DATA (MB(2,K),K=1,11)/11*24/,MB(1,1)/25/,MB(1,2)/27/,
     CMB(1,3)/25/,MB(1,4)/27/,MB(1,5)/25/,MB(1,6)/27/
      DATA MB(1,7)/25/,MB(1,8)/28/,MB(1,9)/24/,MB(1,10)/29/,
     CMB(1,11)/26/
      END
C
C END OF CODE

STD DEVIATION = 0.46362E-06
VARIANCE = 0.21495E-12
```

APPENDIX F

This appendix lists ASM-86 code for executing the filter examined by this report. The execution of this filter would require 14,630 clock cycles, using the figures from Appendix B. Using a 5 MHz clock, the filter execution frequency is 341.76 Hz. The output delay is 5,071 clock cycles, 1.014 milliseconds.

The variable names used in the code relate to the filter coefficients in the following manner:

$$R = r$$

$$A01,\ldots,A011 = a_{01},\ldots,a_{011}$$

Y1 = the output of filter section 1

Y = the filter output

X = the filter input

$$B11,\ldots,B111 = (-b_{11}),\ldots,(-b_{111})$$

etcetera.

The 2P filter form is used (the equation can be found in Chapter IV).

The filter code is

```
IN AX,PORT1
MOV X,AX
FILD X
FLD A01
FMUL ST,ST(1)
FADD T1
FST Y1
```

```
FLD A02
FMUL ST,ST(2)
FADD T2
FST Y2
FADD
```

there are ten sections like
the ones above and below

```
FLD A011
FMUL ST,ST(2)
FADD T11
FST Y11
FADD
```

```
FLD R
FMUL ST,ST(2)
FADD
FISTP Y
MOV AX,Y
OUT PORT2,AX
```

```
FLD A11
FMUL ST,ST(1)
FADD W1
FLD Y1
FLD B1
FMUL ST,ST(1)
FADD ST,ST(2)
FSTP T1
FMUL B21
FSTP W1
FSTP ST(0)
```

there are ten sections
like the one above

```
FMUL A111
FADD W11
FLD Y11
FLD B11
FMUL ST,ST(1)
FADD ST,ST(2)
FSTP T11
FMUL B211
FSTP W11
FSTP ST(0)
```

This concludes the filter code.

REFERENCES

1. Paul Moroney. Issues in the Implementation of Digital Feedback Compensators. Cambridge: MIT Press, 1983.

2. R.E. Rink and H.Y. Chong, "Performance of State Regulator Systems with Floating-Point Computation," IEEE Trans. Automatic Control, Vol. AC-24, No. 3, June 1979, pp. 411-421.

3. L.R. Rabiner, J.W. Cooley, H.D. Helms, L.B. Jackson, J.F. Kaiser, C.M. Rader, R.W. Schafer, K. Steiglitz, and C.J. Weinstein. "Terminology in Digital Signal Processing," IEEE Trans. Audio Electroacoust., Vol. AU-20, No. 5, December 1972, pp. 322-337. Reprinted by IEEE in [27].

4. Alan V. Oppenheim and Ronald W. Schafer. Digital Signal Processing. Englewood Cliffs, N.J.: Prentice-Hall, 1975.

5. Leland B. Jackson, "Roundoff - Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form," IEEE Trans. Audio Electroacoust., Vol. AU-18, No. 2, June 1970, pp. 107-122. Reprinted by IEEE in [28].

6. J.F. Kaiser, "Some Practical Considerations in the Realization of Linear Digital Filters," Proc. 3rd Annual Allerton Conf. Circuit and System Theory, 1965. Reprinted by IEEE in [28].

7. J.B. Knowles and E.M. Olcayto, "Coefficient Accuracy and Digital Filter Response," IEEE Trans. Circuit Theory, Vol. CT-15, No. 2, March 1968, pp. 31-41. Reprinted by IEEE in [27].

8. R.E. Crochiere and A.V. Oppenheim, "Analysis of Linear Digital Networks," IEEE Proc., Vol. 63, No. 4, April 1975, pp. 581-595. Reprinted by IEEE in [27].

9. Bede Liu, "Effect of Finite Word Length on the Accuracy of Digital Filters - a Review," IEEE Trans. Circuit Theory, Vol. CT-18, No. 6, Nov. 1971, pp. 670-677. Reprinted by IEEE in [28].

10. Kirk E. Davis, "Floating-Point A/D and D/A Converter," Patent Application No. 243, 985, March 16, 1981. Available as report no. AD008559 from National Technical Information Service.

11. Ernst Avenhaus, "On the Design of Digital Filters with Coefficients of Limited Word Length," IEEE Trans. Audio Electroacoust., Vol. AU-20, No. 3, Aug. 1972, pp. 206-212. Reprinted by IEEE in [27].

12. R.E. Crochiere, "A New Statistical Approach to the Coefficient Word Length Problem for Digital Filters," IEEE Trans. Circuits and Systems, Vol. CAS-22, No. 3, March 1975, pp. 190-196. Reprinted by IEEE in [27].

13. P.M. Ebert, J.E. Mazo and M.G. Taylor, "Overflow Oscillations in Digital Filters," Bell System Tech. J., Vol. 48, No. 9, Nov. 1968, pp. 3021-3030. Reprinted by IEEE in [28].

14. I.W. Sandberg, "Floating-Point-Roundoff Accumulation in Digital-Filter Realizations," Bell System Tech. J., Vol. 46, No. 8, Oct. 1967, pp. 1775-1791.

15. Bede Liu and Toyohisa Kaneko, "Error Analysis of Digital Filters Realized with Floating-Point Arithmetic," IEEE Proc., Vol. 57, No. 10, Oct. 1969, pp. 1735-1747.

16. Edwin P.F. Kan and J.W. Aggarwal, "Error Analysis of Digital Filter Employing Floating-Point Arithmetic," IEEE Trans. Theory, Vol. CT-18, No. 6, Nov. 1971, pp. 678-686.

17. Toyohisa Kaneko, "Limit-Cycle Oscillations in Floating-Point Digital Filters," IEEE Trans. Audio Electroacoust., Vol. AU-21, No. 2, April 1973, pp. 100-106. Reprinted by IEEE in [27].

18. Russel Rector and George Alexy. The 8086 Book. Berkeley: Osborne/McGraw-Hill, 1980.

19. Stephen P. Morse. The 8086 Primer. Rochelle Park, N.J.: Hayden, 1980.

20. Christopher L. Morgan and Mitchell Waite. 8086/8088 16-Bit Microprocessor Primer. Peterborough, N.H.: Byte/McGraw-Hill, 1982.

21. Intel Corp. The 8086 Family User's Manual. Santa Clara, CA: Intel, 1980.

22. Intel Corp. The 8086 Family User's Manual Numerics Supplement. Santa Clara, CA.: Intel, 1980.

23. Intel Corp. ASM-86 Language Reference Manual. Santa Clara, CA.: Intel, 1982.

24. Intel Corp. Getting Started with the Numeric Data Processor, Application Note AP-113. Santa Clara, CA.: Intel, 1981.

25. R.M. Golden and J.F. Kaiser, "Design of Wideband Sampled-Data Filters," Bell System Tech. J., Vol. 43, No. 4, July, 1964, pp. 1533-1546.

26. George R. Cooper and Clare D. McGillem. Probabilistic Methods of Signal and System Analysis. New York: Holt, Rinehart and Winston, 1971.

27. Digital Signal Processing Committee. Papers in Digital Signal Processing II. New York: IEEE Press, 1975.

28. L.R. Rabiner and C.M. Rader. Digital Signal Processing. New York: IEEE Press, 1972.